МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Изучение деревьев

Вариант 14-D

Студент гр. 8304 Сани З. Б.
Преподаватель Фирсов М.А

Санкт-Петербург
2019

Цель работы.

Получить опыт работы с бинарным деревом реализовать его с помощью динамической памяти.

Постановка задачи.

Бинарное дерево называется бинарным деревом поиска, если для каждого его узла справедливо: все элементы правого поддерева больше этого узла, а все элементы левого поддерева – меньше этого узла.

Бинарное дерево называется пирамидой, если для каждого его узла справедливо: значения всех потомков этого узла не больше, чем значение узла.

Для заданного бинарного дерева с числовым типом элементов определить, является ли оно бинарным деревом поиска и является ли оно пирамидой.

Описание алгоритма.

Для начала программа должна считать данные и занести их в структуру. Дерево реализуется на основе списка: в структуре каждого узла должен храниться 3 поля – data, left, right. data – значение хранящееся в данном узле, left и right – правое и левое поддерево соответственно.

Для определения является ли оно бинарным деревом поиска, мы создали функцию, которая рекурсивно вызывает себя Для каждого узла в дереве и возвращает true, если левый и правый узел удовлетворяют BST, т. е. данные левого узла должны быть меньше данных узла, а данные правого узла должны быть больше данных узла.

Для определения является ли оно пирамидой, мы создали функцию, которая рекурсивно вызывает себя Для каждого узла в дереве и возвращает true, если узел и оба его потомка удовлетворяют пирамиде, т. е. значения всех потомков этого узла не больше, чем значение узла

Спецификация программы.

Программа предназначена для определения того, является ли данное дерево бинарным деревом поиска или пирамидой.

Программа написана на языке C++.

Описание функций и структур данных.

```
bool startTree(std::string const& inputString);
```

Функция принимает входную строку и вызывает bool isCorrectStr(std::string const& inputString); чтобы проверить, является ли строка правильным представлением скобки бианарного дерева ,а затем вызывает функцию, которая строит бианарное дерево constrcTree.

```
bool constrcTree(std::string const& inputString, std::shared_ptr<Node> const& rootPointer);
```

Для составления данных из полученной строки используется рекурсивная функция constrcTree, принимающая рассматриваемый узел и подстроку исходной строки, содержащую данные, которые необходимо внести в рассматриваемый узел. Заполнение реализовано при помощи нескольких циклов while и рекурсивного вызова функции constrcTree для левого и правого поддерева.

```
bool isBST(std::shared_ptr<Node> const& root,std::shared_ptr<Node> const& l=nullptr,
std::shared_ptr<Node> const& r=nullptr);
```

Для определения является ли оно бинарным деревом поиска, isBST принимает корневой узел и его потомков, которые по умолчанию nullptr. затем isBST рекурсивно вызывает себя Для каждого узла в дереве и возвращает true, если левый и правый узел удовлетворяют BST, т. е. данные левого узла должны быть меньше данных узла, а данные правого узла должны быть больше данных узла.

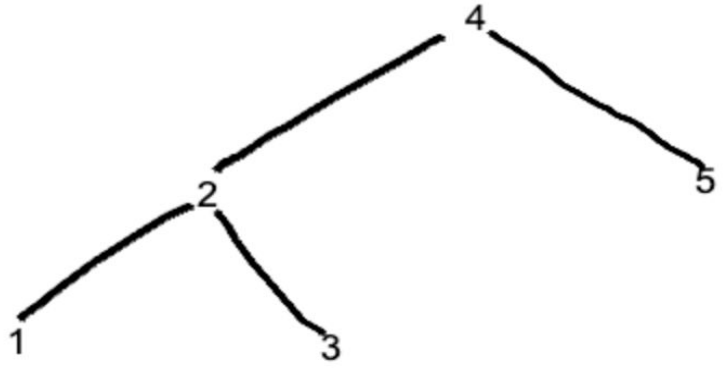
```
bool isPyramid(std::shared_ptr<Node> const& rootPointer);
```

Для определения является ли оно пирамидой, isPyramid принимает корневой узел и рекурсивно вызывает себя Для каждого узла в дереве и возвращает true, если узел и оба его потомка удовлетворяют пирамиде, т. е. значения всех потомков этого узла не больше, чем значение узла.

ПРИЛОЖЕНИЕ

1. Тестирование

Работа программы для выражения (4(2(1)(3))(5))




```
> Choose your input
> 0 - from console
> 1 - from file default file -(default test file is located along the path : test4.txt)
> Any other number to Exit!
0
> Enter binary tree: (4(2(1)(3))(5))
String is correct :)
-->This is a Binary Search Tree

Program ended with exit code: 0
```

Таблица результатов ввода/вывода тестирования программы

Выражение Дерева	Корректность	Определение
(3(2)(4))	String is correct	BST
(4(2)(3))	String is correct	Pyramid
(4(2(1)(3))(5))	String is correct	BST
(6(5(4)(3))(5))	String is correct	Pyramid
(80(75(1)(4))(9(3)(2)))	String is correct	Pyramid
(24(16(8(3)(11))(21))(62(60)(99)))	String is correct	BST
(233(199(33)(56))(200(199)(2)))	String is correct	Pyramid
(4(1(t)(7))(6(q)(2)))	only numbers	-
()	no root element	-
(1(2(4(8)(9))(5(10)(11)))(3(6(0)(13))(7(0)(14))))	String is correct	No definition
(51(36(20(16)(22))(44(37)(50)))(78(66(59)(71))(99(90)(129))))	String is correct	BST
(131(56(44)(3(1)(1)))(17))	String is correct	Pyramid
(19(13(11)(17))(56(30)(78)))	String is correct	BST
(233(199(33)(56))(200(199)(2)))	String is correct	Pyramid
(4(1(0)(7))(6(1)(2)))	String is correct	No definition
(19(13(11)(17))(56(30)(78))))	bracket not balanced	-

```
 test #1 "(3(2)(4))"
String is correct :)
-->This is a Binary Search Tree

test #2 "(4(2)(3))"
String is correct :)
-->This is Pyramid

test #3 "(4(2(1)(3))(5))"
String is correct :)
-->This is a Binary Search Tree

test #4 "(6(5(4)(3))(5))"
String is correct :)
-->This is Pyramid

test #5 "(80(75(1)(4))(9(3)(2)))"
String is correct :)
-->This is Pyramid

test #6 "(24(16(8(3)(11))(21))(62(60)(99)))"
String is correct :)
-->This is a Binary Search Tree

test #7 "(233(199(33)(56))(200(199)(2)))"
String is correct :)
-->This is Pyramid

test #8 "(4(1(t)(7))(6(q)(2)))"
Only Numbers are allowed!
FIX! the above errors and run again!

test #9 "()"
No element was added as root!
FIX! the above errors and run again!

test #10 "(1(2(4(8)(9))(5(10)(11)))(3(6(0)(13))(7(0)(14))))"
String is correct :)
-->None of the above :(


test #11 "(51(36(20(16)(22))(44(37)(50)))(78(66(59)(71))(99(90)(129))))"
String is correct :)
-->This is a Binary Search Tree

test #12 "(131(56(44)(3(1)(1)))(17))"
String is correct :)
-->This is Pyramid

test #13 "(19(13(11)(17))(56(30)(78)))"
String is correct :)
-->This is a Binary Search Tree

test #14 "(233(199(33)(56))(200(199)(2)))"
String is correct :)
-->This is Pyramid
```

```
test #15 "(4(1(0)(7))(6(1)(2)))"
String is correct :)
-->None of the above :(


test #16 "(19(13(11)(17))(56(30)(78))))"
Brackets of the tree is not balanced!
FIX! the above errors and run again!
```

Выводы.

В ходе работе был получен опыт работы с деревом на основе списка.

Исходный код программы

tree.hpp

```
#ifndef tree_hpp
#define tree_hpp

#include <stdio.h>
#include <memory>
#include <iostream>


struct Node
{
    Node() :data(int()),left(nullptr), right(nullptr){}

    int data;
    std::shared_ptr<Node> left;
    std::shared_ptr<Node> right;
};

class MyBinaryTree
{
public:
    explicit MyBinaryTree();
    ~MyBinaryTree() = default;

    std::shared_ptr<Node> rootPointer;

    bool startTree(std::string const& inputString);
    bool construcTree(std::string const& inputString,std::shared_ptr<Node>& rootPointer);
    std::string subString(std::string const& indexString, size_t* stringIndexPointer);
    bool isCorrectStr(std::string const& str);
```

```
    bool isBalanceBracket(std::string const& str);


};


#endif /* tree_hpp */
```

tree.cpp

```
#include "tree.hpp"

MyBinaryTree::MyBinaryTree(){
    rootPointer = std::make_shared<Node>();
}

bool MyBinaryTree::startTree(std::string const& inputString){

    if(isCorrectStr(inputString)){
        construcTree(inputString, rootPointer);
        return true;
    }
    return false;
}

bool MyBinaryTree::construcTree(std::string const& inputString,std::shared_ptr<Node>& rootPointer){
    // the first character is ' ( ' , there is no need to consider it
    size_t stringIndex = 1;

    std::string rootStringValue;
    while (stringIndex < inputString.size() && (inputString[stringIndex] != '(' && inputString[stringIndex]
!= ')'))
    {
        rootStringValue += inputString[stringIndex];
        stringIndex++;
    }
    if (rootStringValue.empty())
        return false;

    rootPointer->data = stoi(rootStringValue);

    //if the end of the line is encountered, the left and right subtree = empty
    if (inputString[stringIndex] == ')')
        return true;

    rootPointer->left  = std::make_shared<Node>();
    std::string bracketsValue = subString(inputString, &stringIndex);

    bool formLeftResult = construcTree(bracketsValue, rootPointer->left);
    if (!formLeftResult)
```

```
        return false;

    //if the end of the line is encountered, the right subtree = empty
    if (inputString[stringIndex] == ')')
        return true;


    rootPointer->right = std::make_shared<Node>();
    bracketsValue = subString(inputString, &stringIndex);

    bool formRightResult = construcTree(bracketsValue, rootPointer->right);
    if (!formRightResult)
        return false;

    //check for correctness of end symbols
    if (inputString[stringIndex] != ')' || stringIndex + 1 != inputString.size())
        return false;

    return true;
}

std::string MyBinaryTree::subString(std::string const& indexString, size_t* stringIndexPointer){
    size_t tmp_ind = *stringIndexPointer;
    int error = 0;
    std::string bracketsValue;

    while (1)
    {
        //write the next character
        bracketsValue += indexString[tmp_ind];
        tmp_ind++;

        if (indexString[tmp_ind] == '(')
            error++;
        if (indexString[tmp_ind] == ')')
            error--;
        if (error < 0)
            break;
    }

    //write )
    bracketsValue += indexString[tmp_ind];
    //shift index for the expression in parentheses to read the second argument
    *stringIndexPointer = tmp_ind + 1;
    return bracketsValue;
}

bool MyBinaryTree::isBalanceBracket(std::string const& str){
    size_t openBracketCnt = 0;
    size_t closeBracketCnt = 0;
```

```
    for (char symb : str){
        if (symb == '(')
            openBracketCnt++;

        if (symb == ')')
            closeBracketCnt++;

        if (closeBracketCnt > openBracketCnt)
            return false;
    }

    return closeBracketCnt == openBracketCnt;
}

bool MyBinaryTree::isCorrectStr(std::string const& str){

//   Checking the input string (tree in a simplified bracket representation) for correctness

    if(!isBalanceBracket(str))
    {
        std::cout << "Brackets of the tree is not balanced!\n";
        return false;

    }

      bool isElem = false;
      bool flagIsCorrect = true;
      if (str[0] != '(') {
          std::cout << "Bracket representation of Binary Tree must start and end with brackets!\n";
          return false;
      }
      if (str[str.length() - 1] != ')') {
         std::cout << "Bracket representation of Binary Tree must start and end with brackets!\n";
          return false;
      }

      size_t indexStart;
      size_t numBrackets = 0;
      for (indexStart = 1; indexStart < str.length() - 1; indexStart++) {

          if (str[indexStart] != ' ' && str[indexStart] != '(' && str[indexStart] != ')') {
              if (!isElem) {
                  while (str[indexStart] != '(' && str[indexStart] != ')') {
                      if(!isdigit(str[indexStart])){
                          std::cout << "Only Numbers are allowed!\n";
                          return false;
                      }
                      indexStart++;
                  }
                  isElem = true;
              }
```

```
        }

        if (str[indexStart] == '(') {
            if (!isElem) {
                std::cout << "No element was added as root!\n";
                return false;
            }
            numBrackets++;
            if (numBrackets > 2) {
                std::cout << "More than 2 branches!\n";
                return false;
            }
            size_t indexEnd = indexStart;
            int openB = 1;
            int closeB = 0;
            while (openB > closeB) {
                indexEnd++;
                if (indexEnd >= str.length()) {
                    std::cout << "Incorrect input!\n";
                    return false;
                }
                if (str[indexEnd] == '(') {
                    openB++;
                }
                else if (str[indexEnd] == ')') {
                    closeB++;
                }
            }
             flagIsCorrect = flagIsCorrect && isCorrectStr(str.substr(indexStart, indexEnd - indexStart + 1));
             indexStart = indexEnd;
        }
    }

    if (str[indexStart] == ')') {
        if (!isElem) {
            std::cout<< "No element was added as root!\n";
            return false;
        }
        if (indexStart == str.length() - 1) {
            return flagIsCorrect;
        }
        else {
            std::cout << "Incorrect input!\n";
            return false;
        }
    }

    return true;
}
```

main.cpp

```
#include <iostream>
#include "tree.hpp"
#include <stdio.h>
#include <fstream>

void defineTree(const std::string &input);
void ReadFromFile(const std::string &filename);
bool isPyramid(std::shared_ptr<Node> root);

void ReadFromFile(const std::string &filename)
{
   std::ifstream file(filename);

   if (file.is_open())
   {
      int count = 0;
      while (!file.eof())
      {
         count++;
         std::string input;
         getline(file, input);
         std::cout << "test #" << count << " \"" + input + "\""<< "\n";
         defineTree(input);
      }
   }
   else
   {
      std::cout << "File not opened!"<<"\n";
   }
}

bool isBST(std::shared_ptr<Node> root, std::shared_ptr<Node> l=nullptr, std::shared_ptr<Node>
r=nullptr)
{
   // Base condition
   if (root == nullptr)
      return true;

   // if left node exist then check it has
   // correct data or not i.e. left node's data
   // should be less than root's data
   if (l != nullptr and root->data <= l->data)
      return false;

   // if right node exist then check it has
   // correct data or not i.e. right node's data
   // should be greater than root's data
   if (r != nullptr and root->data >= r->data)
      return false;
```

```
    // check recursively for every node.
    return isBST(root->left, l, root) and isBST(root->right, root, r);
}

bool isPyramid(std::shared_ptr<Node> root)
{
    /* left_data is left child data and
       right_data is for right child data*/
    int left_data = 0, right_data = 0;

    /* If node is NULL or it's a leaf node
    then return true */
    if(root == nullptr || (root->left == nullptr && root->right == nullptr))
        return true;
    else
    {
        /* If left child is not present then 0
        is used as data of left child */
        if(root->left != nullptr)
            left_data = root->left->data;

        /* If right child is not present then 0
        is used as data of right child */
        if(root->right != nullptr)
            right_data = root->right->data;

        /* if the node and both of its children
        satisfy the pyramid return true else false*/
        int childrenMax = left_data > right_data ? left_data : right_data;
        if((root->data > childrenMax) && isPyramid(root->left) && isPyramid(root->right))
            return true;
        else
            return false;
    }
}


void defineTree(const std::string &input){
    MyBinaryTree tree;

    if(tree.startTree(input)){
        std::cout<<"String is correct :)\n";
        if (isBST(tree.rootPointer))
              std::cout << "-->This is a Binary Search Tree"<<"\n\n";

          else if(isPyramid(tree.rootPointer))
              std::cout << "-->This is Pyramid"<<"\n\n";

          else
              std::cout << "-->None of the above :(\n"<<"\n\n";
    }else{
```

```
        std::cout<<"FIX! the above errors and run again!\n\n";
    }
}

int main() {
    std::cout << "> Choose your input option" << "\n";
    std::cout << "> 0 - from console" << "\n";
    std::cout << "> 1 - from file default file -(default test file is located along the path : test4.txt)" << "\n";
    std::cout << "> Any other number to Exit!" <<"\n";

    int command = 0;
    std::cin >> command;
    std::cin.ignore();

    switch (command)
    {
        case 0:
        {
            std::cout << "> Enter binary tree: ";
            std::string input;
            std::getline(std::cin, input);
            defineTree(input);
            break;
        }
    case 1:
        {
            std::cout << "> FilePath:\n ";
            std::string filePath;
            std::cin >> filePath;
            ReadFromFile(filePath);
            break;
        }
    default:
            std::cout << "GOODBYE!\n";
    }

    return 0;
}
```